(Project Narrative)

Functorial Dynamics and Interaction: A computational design environment

David I. Spivak

Abstract

Frederick A. Leve	— Dynamics and Control
Richard D. Riecken	— Science of Information, Computation, Learning, and Fusion
Tristan Nguyen	— Information Assurance and Cybersecurity

(*Abstract may be publicly released*)

We propose a new, synthetic theory of context-dependent interaction that is fully computational and compositional. The interacting systems include traditional dynamical systems, as well as computer programs, resources or agencies in a society, robots, AIs, etc. These systems interact within a given arena by observing each other's behavior and reacting to it, where the reaction from a given system depends on its current *reactive state*. In special cases, the interaction is determined by a fixed signal-passing formula: system A consistently receives information x from system B. Our theory is broader, allowing for context-dependent interaction: the interaction pattern itself can change based on the reactive states of all the systems involved. One can imagine chemical bonds forming or breaking, companies changing suppliers, a robot opening its eyes, a GPS going offline, a memory being activated, etc. In each case, the state of the systems in the arena determine not just the content but also the *sort* of information that flows between them.

Category theory and type theory are especially suited to synthetic, compositional reasoning. We propose the category **Poly** of polynomial functors for the job described above, because it has excellent formal properties, is extremely versatile, and can be easily embedded into a computer language like Idris, with full-blown programming capacity, accessible to a large group of engineers.

What makes **Poly** appropriate and versatile for this work is the fact that its many operations (four interacting monoidal structures, two closure operations, etc.) are all useful in the theory of interacting dynamical systems, as we will show in this proposal. For example, the composition operator $\circ$ allows one to arbitrarily speed up discrete dynamical systems; we can use it to simulate continuous behavior and apply numerical methods to solve differential equations. We discuss how to apply these ideas to provide compositional invariants of interacting systems and resources. The desired result is a mathematically-based, computational design environment, fully-equipped for creating, simulating, and proving properties of context-dependent interactions. With this established, we can pursue our interest in fundamental questions of emergence, e.g. the origins of control and intelligence.

Contents

Statement of Objectives

0 Statement of Objectives

The overall purpose of this work is to *pursue fundamental scientific questions* within a *formal reasoning system*. Like many before us, we are interested in how intelligence arises from non-intelligence, life from non-living matter, language from chemistry, etc. Observing complex behavior emerging from black-box systems, we want to understand what is actually producing it. In many cases of interest, we find the concept of control and influence: the fact that a system somehow tames an otherwise hostile environment.

To pursue these questions, our main objective is to provide a computational and compositional design environment within which to build and study context-dependent interactions between dynamical systems, broadly construed. These systems react to the distinctions they perceive within their arena of interaction, causing them to change their abstract or concrete position—e.g. move from being non-communicative to being communicative, or from the home to the store—at which point they receive new distinctions to react to. But the way that information flows through real systems is very often not fixed: in one moment A is receiving from B, but in the next moment the two are no longer in communication at all. We want to study this situation—which we call *context dependent interaction*—in a compositional way: we should be able to build and analyze complex wholes by putting together previously-built and more simply-analyzed parts. Thus our objective design environment should make it as convenient as possible to work with and reason about these sorts of complex, context-dependent interaction settings.

While often one uses any sort of math available—calculus, probability, temporal logic, computability theory, etc.—here we wish to pursue these questions within a single mathematical theory. Category theory and type theory are amenable to this task because they are both compositional and highly interoperable. In particular, it appears that *a single category* **Poly**, of polynomial functors, is suitable for everything described above, though we will always work in whatever setting is most appropriate to the task at hand.

More specific objectives include the following:

1. Investigate the mathematical theory of polynomial functors;
2. Investigate their application to context-dependent interaction;
3. Produce a design environment and/or software library for this work;
4. Formalize differential equations and their solutions in this setting;
5. Formalize control, stability, feasibility, etc., in this setting;
6. Compute invariants and coarse behavior estimates of interacting systems;
7. Develop notions of hierarchical planning and communication;
8. Develop game theory—including "when to engage in a game"—for agents;
9. Consider synthetic biology and artificial intelligence.

Again, to the greatest extent possible, we hope to do all of this within the single categorical setting of polynomial functors, because doing so will make the theory highly interoperable. While we have reason to hope this will work, we may need to move into a more general categorical theory of dynamical systems. We are open to whatever mathematical scaffolding allows us to pursue the big questions within a formal system.

Research effort

1 Dynamics and interaction

1.1 Arenas and models

We consider a *system* to be anything in the real world for which one has a model in hand. This model—such as the control panel in an airplane cockpit—must be sufficiently aligned with reality that one can operate within the model and, by doing so, be successful in reality [Spi17]. A pilot navigates a plane through a real-life situation using only those aspects of reality that can be made present to the pilot in the cockpit.

Thus a model is a bridge between the outer world and a domain in which we know how to operate; it is a lens by which we can focus our attention on that which we can do something about. The model allows us to:

1. steer, by casting our internal choices as *positions* in the exterior world,
2. watch, by interpreting external conditions as *distinctions* we can make in our interior world.

We refer to both the outer world (e.g. the real-life situation) and the inner world (e.g. the cockpit) as *arenas*. An arena comprises some notion of position—where we are—and some notion of distinction—what we can notice. The word "position" is meant in a very broad sense: our position includes our attitude, our expression, and anything else that can be observed from the outer world. When one sets out a type of positions and, for each position a type of distinctions available in that position, one has defined an arena.

Arenas and the models that connect them form a mathematical *category*. Its objects are arenas, and its morphisms are models. It turns out that this category is equivalent to a well-known category, that of polynomial functors; which we denote **Poly** [GK12] [SM20]. In a surprising way, the simplest polynomials from algebra class encode (the simplest) arenas. To see this, take any polynomial, e.g. $P(y) = y^2 + 2y + 1$, and write it out as a sum $\sum y^i$ of pure-power terms:[1]

$$P(y) = y^2 + y^1 + y^1 + y^0$$

As an arena, the four summands refer to four positions; in the first position there are two available distinctions, in the second and third positions there is only one available distinction, and in the fourth there are none. But these are only the simplest polynomials; in dependent type theory we can be much more general, as one can see in the following Idris code:

```
record Arena where
  constructor MkArena              -- To construct an arena, define:
  position    : Type               -- the meaning of "positions", and
  distinction : position -> Type   -- the meaning of "distinctions".
```

This syntax says that one can choose any type to serve as the positions: one is not limited to the four element set as above, but instead could use the type `Double` of

[1]We use the variable y because it stands for Yoneda; indeed, the fundamental category theoretic result known as the Yoneda lemma is involved when one regards the polynomial $y^2 + 2y + 1$ as a functor. We refer to y as a *functorial variable*. One can think of a polynomial simply as a data structure for an arena.

double-precision floating point numbers for encoding the real numbers $\mathbb{R}$. Similarly one could define the type of positions to be $\mathbb{R}^n$ (`Vect n Double`) for any $n : \mathbb{N}$, but also much more generally one can use `Arena : Type` as its positions. It is as easy to express a machine that spits out arenas as a machine that spits out Doubles.

Here's a more or less down-to-earth example:

```
Lineland : Arena
Lineland = MkArena Double dis  --In Lineland positions are real numbers.
  where
    dis : Double -> Type       --The distinctions may vary by location:
    dis x = if x < 0                  --At negative x's,
            then Double               --we can only hear; otherwise,
            else (Double, Double)     --we can both see and hear.
```

This too is a polynomial; i.e. `Lineland` is an object in **Poly**. The above Idris code is perfectly good as mathematical syntax, though a more traditional polynomial form for the arena called `Lineland` would be

$$\texttt{Lineland} \cong \sum_{\{x:\mathbb{R}|x<0\}} y^{\mathbb{R}} + \sum_{\{x:\mathbb{R}|x\geq 0\}} y^{(\mathbb{R}^2)}.$$

In the category **Poly**, the infinite-exponent terms $y^{\mathbb{R}}$ and $y^{(\mathbb{R}^2)}$ count as polynomials, and their infinite sum `Lineland` is also a polynomial. The main point here is that arenas are fully general: the positions or distinctions can be of any type whatsoever, much more complex than `Lineland`. When reading this proposal, the terms *polynomial* and *arena* should thus invoke a sense of broad scope, in keeping with the definition of arena above.

The morphisms in **Poly** are what we called models at the start of Section 1.1. Mathematically, they are precisely the *natural transformations* between polynomial functors. But here is an equivalent definition that fits our informal description above.

```
record Model (internal : Arena) (external : Arena) where
  constructor MkModel
  steer : position internal -> position external
  watch : (p : position internal) ->
          distinction external (steer p) ->
          distinction internal p
```

What we call polynomials are also called *containers* in computer science ([AAG03], [AAG05]). They were called *dependent lenses* in [Spi19], and are a generalization of lenses in the sense of [BPV06].

A three-way bridge. From this point on, the reader can consider the presentation in any of three ways:

1. in the intuitive terms of arenas and models,
2. in the mathematical terms of the category **Poly**, or
3. in the computational terms of a dependently-typed programming language.

When we speak in one, we are speaking in all three—we will only make intuitive statements when they are backed up by the mathematical formalism of **Poly**. Similarly, any statement we make about **Poly** is a statement about arenas and models.

Before discussing what we can do within this setup, let's step back and consider our purpose and how we plan to pursue it.

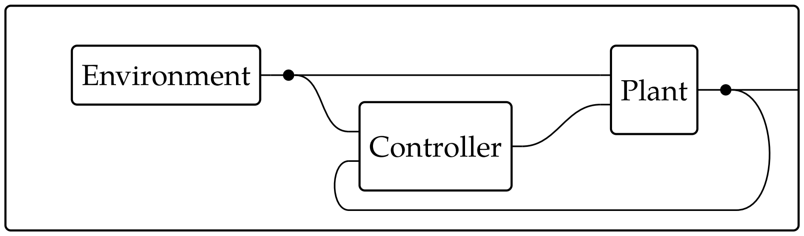
1.2 Purpose and objectives

Our overall purpose is to pose and investigate fundamental scientific questions, for example to understand the nature of intelligence, and how it arises from the interactions of non-intelligent resources; and to understand life and autopoiesis, and how they emerge from the interactions of non-living matter. What sorts of internal operations allow a given black box system to do what it does, to survive within adversity? What allows these methods to scale to the real world? How does it know what to do, and who to interact with, in order to achieve its goals, and from where do these goals arise? In particular, we aim to understand the notion of *control*: what it means abstractly, and what makes one form of control ultimately more successful and beneficial to the systems involved.

Tackling these questions may be a never-ending pursuit, but it guides our work. In order to approach these questions mathematically, we must first express them formally and then reason about them computationally. We will show that the category **Poly** has a wonderful combination of being very expressive, while also being completely computational.

Expressive power. Below, we will show that within **Poly**, one can express the following:

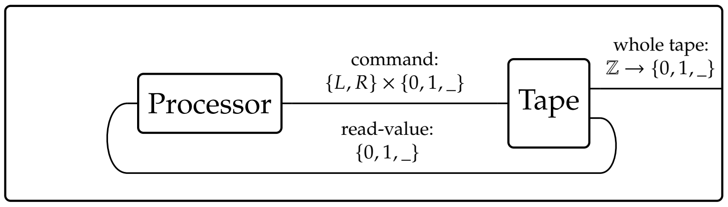
- Open discrete dynamical systems—machines that take a time-varying input signal, change their state accordingly, and produce output.
- Wiring diagrams, such as this one:



(1)

- Changing or *context-dependent* communication patterns, where the type of data flowing between systems can change based on internal and environmental contexts, [ST17].
- Coarse-graining of system interfaces.
- Planning and strategy, in the sense of games.
- Non-deterministic behavior, rewards, exceptions, etc.
- Numerical approximations of continuous dynamical system behavior.

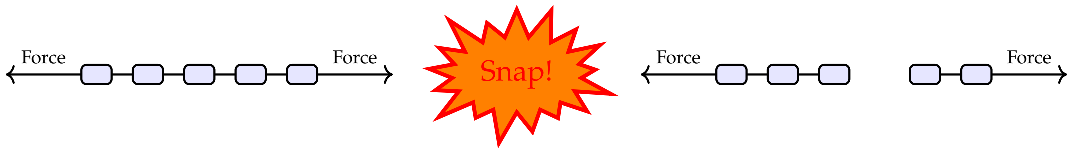
We feel compelled to reiterate that this is much more general than usual considerations. For example, without even allowing for context-dependent interactions we can already encode all Turing machines, which are dynamical systems with the following wiring pattern:


Processor
command: $\{L,R\}\times\{0,1,\_\}$
read-value: $\{0,1,\_\}$
Tape
whole tape: $\mathbb{Z}\to\{0,1,\_\}$

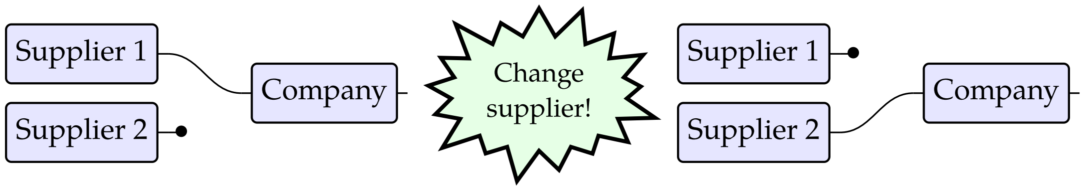
Here a central processor sends commands—such as "write to tape", or "move left"—to the infinite tape, which in turn follows the command and reports the value of the current read-head, 0, 1, or blank.

But the real world is more complex, because the interaction pattern—how information flows between systems—is not fixed, but varies with the reactive states of the systems involved. For example:

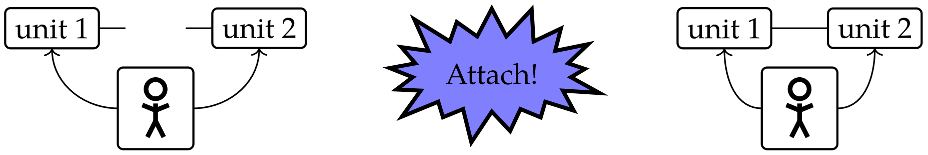
1. Airplanes only communicate when they are within a certain range;
2. A phone is connected to 4G or to wifi depending on circumstances;
3. A person can choose when to open (receive input through) their eyes and when to speak (produce output);
4. Memories (or K-lines) can be active or inactive, communicating with other resources or not, based on complex conditions;
5. When too much force is applied to a material, bonds can break;



6. A company may change its supplier at any time;



7. When someone assembles a machine, their own outputs dictate the connection pattern of the machine's components.



In each case the wiring diagram—the connection pattern—changes based on the states (position, decision-making, environmental context, etc.) of some or all of the systems involved.

We hope it is clear that the language **Poly** is extraordinary in its expressivity. We are not just talking about expressing the above ideas "formally", or mathematically, or even within the sub-discipline of category theory; we are expressing them all within one specific category.

Computational power. This language was designed to compute. Every aspect of this theory takes place in the purely formal and fully computational language of arenas and models, e.g. as embedded in Idris. All the connection patterns between dynamical systems are stored as computations that can be performed at any time.

To explain this, we begin by looking at the basic structures that make traditional control theory possible. Namely, the various well-interacting structures that exist on the set $\mathbb{R}$ of real numbers.

The set $\mathbb{R}$ of real numbers has the following structures and properties:

1. $\mathbb{R}$ is a field: one has the constants and operations $0, +, -, 1, \times, \div$.
2. $\mathbb{R}$ is a linear order: one can compare reals using $\leq$.
3. $\mathbb{R}$ is a complete metric space: there is a notion of distance between reals.

These structures are all useful in modeling dynamics and control. We can use them to express notions like rates of change, and form differential equations with them, which in turn express very complex real-world situations. Importantly, these structures all come with computational counterparts: we can compute $\pi * \pi$ to arbitrary precision, decide in finite time which is bigger e^{π} or π^{e}, etc.

The category **Poly**—whose objects are arenas and whose morphisms are models—also comes with many useful structures. While some category theory is required to understand the technical detail, we hope non-category theorists will see the parallel:

Theorem 1.3. The category **Poly** has the following structures:

1. **Poly** has coproducts and products: $0, +, 1, \times$.
2. **Poly** has two additional monoidal structures: $\otimes$ and $\circ$.
3. **Poly** has two closure (internal hom) operators: one for $\times$ and one for $\otimes$.
4. **Poly** is a distributive category in two ways and also a duoidal category.
5. **Poly** has an orthogonal "vertical/cartesian" factorization system.
6. **Poly** admits an adjoint quadruple with **Set**.

The parallel is that, like for the set $\mathbb{R}$ of real numbers, **Poly** has many formal operations at its disposal. Furthermore, the functors **Set** $\to$ **Poly** allow us to encode the set $\mathbb{R}$ of real numbers *within* **Poly**, so all of $\mathbb{R}$'s properties as a field etc. can be used as well.

1.4 Translation: using Poly operations in practice

Our first order of business is to clarify how to model dynamical systems within **Poly**. To begin, everything we discuss will be about a (context-dependent) generalization of discrete dynamical systems. A reader eager for more modeling power might note that we often want to consider variants of this: continuous or hybrid dynamical systems, non-determinism, etc. We will get to all that in due time; let's begin with the basics.

Define an *input-output arena* to be one where the distinctions do not depend on the positions: we think of positions as outputs and distinctions as inputs.[2] In Idris, input-output arenas can be defined as follows:

```
IOArena : Type -> Type -> Arena                -- same distinctions
IOArena inp outp = MkArena outp (\_ => inp)    -- regardless of position
```

In terms of polynomials, input-output arenas are exactly the monomials By^A, where A is the set of inputs and B is the set of outputs. The simplest arena is the closed box: unchanging input and unchanging output. As a polynomial, it is just y; as a box it looks like a simple enclosure $\square$; and as an arena it is `IOArena () ()`.

A central definition for dynamics is what we call a *self-referencer*. These are arenas whose positions and distinctions have the same type; we refer to both the positions and distinctions as *states*. In Idris we define `Self states = IOArena states states`. In **Poly** these are monomials of the form Sy^S for some set S. We can now state the following proposition.

Proposition 1.5. Let S, A, B be sets. The following are equivalent:

1. A stream processor that takes streams of A's and converts them to streams of B's by updating an internal state variable of type S;
2. A (possibly infinite-state) Moore machine with states S, inputs A, and outputs B;
3. A pair of functions *readout* : $S \to B$ and *update* : $S \times A \to S$;
4. An F-coalgebra $S \to F(S)$, where $F(Y) \coloneqq B \times Y^A$ for any set Y;
5. A natural transformation of polynomial functors $Sy^S \to By^A$;
6. A model (in the sense of the definition on page 7) of `Self S` in `IOArena A B`.

This is clearly an idea with many formalisms, but there are important advantages to working with the last two, i.e. completely within **Poly**. First, by doing so we can immediately generalize this idea beyond monomials, i.e. beyond input-output arenas, so that a system may have more distinctions available in one position than in another. It is in this way that we may model the seven sorts of context-dependent interactions (breaking bonds, changing suppliers, etc.) from page 9. We will see this in Example 2.4.

Let us summarize Proposition 1.5 in language we will use throughout the rest of the proposal.

Definition 1.6 (Dynamical system)**.** A (context-dependent) dynamical system comprises:

1. a type s, called its *states*,
2. an arena I : **Poly**, called its *interface*, and
3. a morphism $sy^s \to I$, called its *dynamics*.

```
record DynamicalSystem where          --A dynamical system
  constructor MkDyn                   --is constructed by choosing
```

[2]To think of outputs in terms of positions, consider the act of talking (giving outputs). In that case, the quickly-changing *positions* of ones mouth and larynx are sufficient to "output" the desired expression.

```
    States    : Type                --any Idris type as its states,
    Interface : Arena               --any Arena as its interface, and
    Dynamics  : Model (Self State) Interface-- any Model as its dynamics.
```

Just as importantly, by working within **Poly**, rather than in the other formalisms in Proposition 1.5, one has a huge wealth of mathematical structure at one's disposal. Below we will see how to employ many of these structures. For example, one can express wiring diagrams and more general interaction schemes. One can also perform various operations on systems, including speeding them up arbitrarily, by which we can perform numerical integration of continuous dynamical systems expressed as ODEs. These systems come equipped with a natural game semantics, which may be useful in learning, e.g. Monte Carlo tree search style algorithms. Every arena A has a natural complement (denoted $[A, y]$), which is the arena of its universal controller; i.e. $[A, y]$ outputs exactly the sort of values that A inputs, and inputs exactly the sort of values that A outputs. Finally, we can express nondeterminism, exceptions, rewards, etc., using (co-) monads.

2 Available operations and their meaning

We will now explain how the operations discussed in Theorem 1.3 make **Poly** an apt language in which to work with dynamical systems.

It's important to think of polynomials as data structures, not as functions. They are arenas of interaction, and so when we add or multiply them, we are manipulating arenas.

Sums. Let's begin with some simple arenas like $A = y^3 + 2y$ and $B = 4y^2 + y + 1$, which have only a few positions (three and six, respectively) and few distinctions in each. Their sum,

$$A + B \cong y^3 + 4y^2 + 3y + 1$$

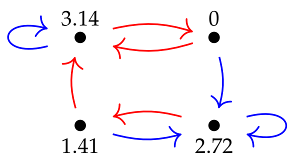
represents an arena in which there are nine positions: each position is either a position in A or a position in B. Similarly, the distinctions available are as they were in A or B. If you own two houses, you can choose where to be in either one or the other, and you'll receive the distinctions available there: this is what adding polynomials means.

Products. The product of polynomials, again let's say $A = y^3 + 2y$ and $B = 4y^2 + y + 1$

$$A \times B \cong 4y^5 + y^4 + 9y^3 + 2y^2 + 2y$$

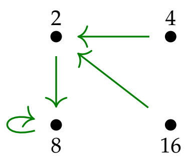
is quite meaningful from a dynamical systems point of view, as we now show in a simple example.

Example 2.1. Let's consider a four-state dynamical system that takes inputs $\{r, b\}$ and gives outputs in $\mathbb{R}$. In other words, this is a morphism $4y^4 \to \mathbb{R}y^{\{r,b\}}$, and we can draw any such morphism quite simply as a labeled transition system:

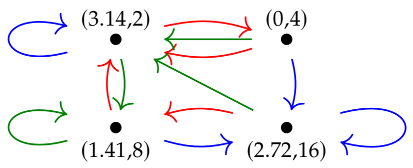


Each bullet refers to a state. It is labeled by its output position in $\mathbb{R}$, and it has exactly one red arrow and one blue arrow emanating from it, indicating how that state is updated upon encountering an r or a b distinction. Of course, one can imagine the exponent being much larger than $\{r, b\}$, e.g. something like **String**, where instead every string would cause a change of state.

Suppose we also have a dynamical system with the same states and outputs (four states, outputs in $\mathbb{R}$), but now it has makes one distinction $\{g\}$. It is a morphism $4y^4 \to \mathbb{R}y^{\{g\}}$, and again we draw one such morphism as a labeled transition system:



No matter how we choose these labeled transition systems, the *universal property of products* says that we automatically get a unique way to put them together. We obtain a morphism $4y^4 \to (\mathbb{R}y^{\{r,b\}} \times \mathbb{R}y^{\{g\}})$, which is a new four-state dynamical system, this time in the arena $(\mathbb{R}^2)y^{\{r,b,g\}}$. With the examples above, it looks like this:



Thus the intuitively obvious act of overlaying these dynamical systems falls out of the mathematics, in particular the universal property of products $\times$ in **Poly**. This works for non-monomial (context-dependent) interfaces as well.

Dirichlet products. The tensor, or *Dirichlet* product of polynomials is denoted $\otimes$. It may look uncanny at first because it is almost the same as $\times$, except that one multiplies exponents rather than adding them. For example if $A = y^3 + 2y^1$ and $B = 4y^2 + y^1 + y^0$, then we have

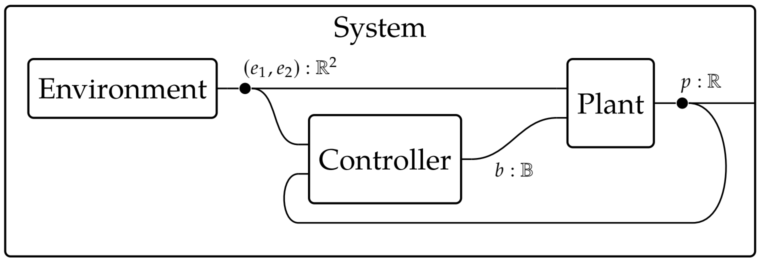
$$\begin{aligned} A \otimes B &\cong 4y^{3*2} + y^{3*1} + y^{3*0} + 8y^{1*2} + 2y^{1*1} + 2y^{1*0} \\ &\cong 4y^6 + y^3 + 8y^2 + 2y + 3 \end{aligned}$$

Dirichlet product is arguably the most important operation in **Poly** from the viewpoint of interacting dynamical systems, and it seems to have been overlooked in the literature on coalgebras and dynamical systems. We have the following:

Proposition 2.2. Context-dependent dynamical systems (Definition 1.6) are closed under $\otimes$.

Proof. The Dirichlet product of self-referencers sy^s and ty^t is again a self-referencer, sty^{st}. Thus if $sy^s \to A$ and $ty^t \to B$ are dynamical systems, their Dirichlet product, $sty^{st} \to A \otimes B$ is too. It has states $s \times t$. $\square$

Example 2.3 (Wiring diagrams). We redraw the diagram from (1), this time labeling each wire with the type of information it carries.



(2)

Here $\mathbb{B}$ is the type of booleans (e.g. on or off), though of course one can replace it with any type whatsoever. The three internal boxes and one external box in the diagram correspond to polynomials

$$E = \mathbb{R}^2 y \qquad C = \mathbb{B}y^{\mathbb{R}^3} \qquad P = \mathbb{R}y^{(c\mathbb{R}^2)} \qquad S = \mathbb{R}y$$

Indeed as we mentioned on page 11, input-output arenas with input A and output B correspond to monomials By^A. Since the controller outputs $\mathbb{B}$ and inputs $\mathbb{R}^2 \times \mathbb{R}$, it corresponds to the monomial $\mathbb{B}y^{\mathbb{R}^3}$.

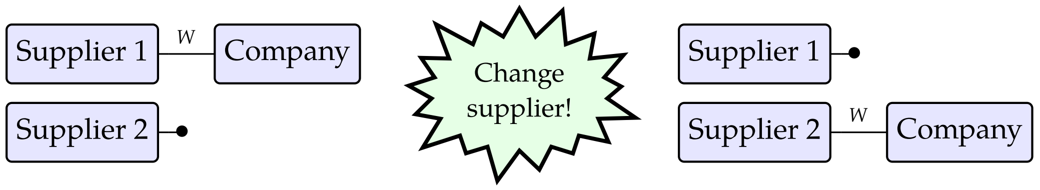
The wiring diagram (2) itself is uniquely represented as a model, or morphism of polynomials $E \otimes C \otimes P \to S$, as defined on page 7. It consists of two functions

$$\begin{array}{cc} \mathbb{R}^2 \times \mathbb{B} \times \mathbb{R} \to \mathbb{R} & \qquad \mathbb{R}^2 \times \mathbb{B} \times \mathbb{R} \to 1 \times \mathbb{R}^3 \times \mathbb{B} \times \mathbb{R}^2 \\ ((e_1, e_2), b, p) \mapsto p & \qquad ((e_1, e_2), b, p) \mapsto (1, (e_1, e_2, p), b, (e_1, e_2)) \end{array}$$

These just say how the signals move around the wiring diagram (2). The first says that what's read out of the entire system is p, the plant temperature. The second says that the environment passes its variables to both the controller and the plant, the controller passes its variables to the plant, and the plant passes its variable to the controller. We've algorithmically encoded the wiring diagram (2) itself as a morphism in **Poly**.

All the polynomials shown in Example 2.3 are monomials, and the maps imply a fixed wiring shape; but next we see that this can be generalized to context-dependent interactions.

Example 2.4. Consider the example shown in Section 1.2, redrawn and simplified a bit:



The reason for redrawing the picture is that we wish to emphasize that what's "causing" the change of supplier is a change in the company's *position*. We have also simplified by assuming that the output of the company is trivial, just to bring out the key point: context-dependent wiring.

The type of each supplier is Wy: it outputs widgets. The type of the company is $2y^W$: it outputs an element in $\{1, 2\}$, indicating which supplier it wants, and it inputs a widget. The information flow pattern shown is a context-dependent wiring diagram $(Wy) \otimes (Wy) \otimes (2y^W) \to y$, in other words a morphism $2W^2y^W \to y$ in **Poly**. When one traces through what this means, it is given by a function $2W^2 \to W$, namely the evaluation map.

We include these details so that expert readers can see how **Poly** straightforwardly encodes the context-dependent wiring diagrams as claimed in Section 1.2.

Internal homs. One does not usually think it is possible to exponentiate polynomials, i.e. to raise P to the Q power and get another polynomial, but it makes sense from a category-theoretic point of view. Recall that in numbers, A^B counts the number of functions from a set with B elements to a set with A elements, and there is a universal evaluation map $B \times A^B \to A$. Similarly in polynomials, P^Q tells us the ways to make a model of arena Q in arena P. In particular, there is a universal evaluation map $Q \times P^Q \to P$. We will not give the formula here (see [SM20]), but instead say how it can be used.

Suppose one has a dynamical system $sy^s \to Q$ with interface Q, but in order for it to interact with other dynamical systems, we wish it had interface P. One method to fulfill the wish is to have a model $Q \to P$; then we can compose to get a dynamical system $sy^s \to Q \to P$. Another method uses exponentiation. Namely, if we could find a suitable system $sy^s \to P^Q$, we could combine the two as in Example 2.1 to get $sy^s \to Q \times P^Q$ and then evaluate to get $sy^s \to P$ as desired.

Just like $\times$ has a corresponding notion of exponentiation and a universal evaluation map, so does $\otimes$. These are both called *monoidal closed structures*; we denote the one for $\otimes$ by $[Q, P]$ rather than P^Q. The universal evaluation is a morphism $Q \times [Q, P] \to P$.

One can see the polynomial $[P, y]$ as the universal control interface for P. A position of $[P, y]$ is precisely a function f that takes every position of P and provides a distinction there: it feeds P something of the type it needs. Given such a function, the distinctions available there in $[P, y]$ are precisely the positions of P, i.e. its outputs. So $[P, y]$ provides what P requires, and it requires what P provides. Moreover, any other interface Q that interacts with P in a closed system, i.e. for which there is a map $P \otimes Q \to y$, induces a unique map $Q \to [P, y]$. Thus $[P, y]$ is the universal control interface for P.

Coarse graining. We may sometimes wish to coarse-grain the arena of a dynamical system, in order to simplify our analysis. That is, we partition the positions or outputs of the system into broader classes, in order to leave aside some details in our analysis. And for such an analysis we may also want to assume that the inputs to the system will always land in some more specific range.

In terms of arenas, this is a very natural thing to do. It is given by a map (an epimorphism) of arenas $A \twoheadrightarrow A'$.

Proposition 2.5. Let $A = (pos, dis)$ and $A' = (pos', dis')$ be arenas. A morphism between them is an epimorphism iff the steering function $f : pos \twoheadrightarrow pos'$ is surjective and, for each position $p : pos$ the watching function $dis'(f p) \rightarrowtail dis(p)$ is injective.

Thus we see that epimorphisms in **Poly** make it natural to coarse-grain systems.

Composition product. Any two polynomials can be composed; for example if $A = y^3+1$ and $B = 4y^2 + y$ then

$$B \circ A \quad = \quad 4A^2 + A \quad \cong \quad 4y^6 + 9y^3 + 5.$$

This operator again has interesting semantic meaning for dynamical systems, in terms of multi-move strategies. Composition product will also let us define what it means to "speed up" a dynamical system.

To explain this, it will help to rewrite our polynomials very explicitly as a sum followed by a product. Choose any polynomial P, thought of as an arena with positions $\texttt{pos}$ and distinctions $\texttt{dis}(p)$. For each $p \in \texttt{pos}$, we have

$$P = \sum_{p:\texttt{pos}} \prod_{d:\texttt{dis(p)}} y$$

One can read this as "a choice of position p and, for every distinction d available at p, I have a y", where the functorial variable y should be thought of as "unknown future". Thus, for example, $P \circ P \circ P$ would be the following polynomial:

$$P \circ P \circ P \cong \sum_{p_1:\texttt{pos}} \prod_{d_1:\texttt{dis}(p_1)} \sum_{p_2:\texttt{pos}} \prod_{d_2:\texttt{dis}(p_2)} \sum_{p_3:\texttt{pos}} \prod_{d_3:\texttt{dis}(p_3)} y$$

One can read this as "a choice of position p_1 and, for every distinction d_1 available at p_1, a choice of position p_2 and, for every distinction d_2 available at p_2, a choice of position p_3 and, for every distinction d_3 available at p_3, I have a y."

We recognize this as a strategy: "I'll do p_1, and for whatever move d_1 she makes, I'll do p_2, ..." Thus $P^{\circ \ell} = P \circ P \circ \cdots \circ P$ denotes the set of length-ℓ strategies.

Next we see how to speed up a dynamical system using composition product.

Proposition 2.6. Self-referencers $S = sy^s$ are $\circ$-comonoids. That is, there is a canonical map $S \to S^{\circ \ell}$ for any $\ell \in \mathbb{N}$.

It follows that if $f : sy^s \to P$ is any dynamical system, then there is an induced dynamical system with interface $P^{\circ \ell}$, namely the composite

$$S \to S^{\circ \ell} \xrightarrow{f^{\circ \ell}} P^{\circ \ell}. \tag{3}$$

Eq. (3) says we may take any dynamical system, even one whose input type depends on its last output, and speed it up by an arbitrary factor $\ell \in \mathbb{N}$: in a single time step, the new machine produces ℓ outputs and takes in ℓ inputs of the form specified by the original dynamical system.

Example 2.7 (Differential equations). Consider the system of ODEs $\dot{x} = f(x)$ where x and f are n-dimensional vectors; we take the system to be autonomous for simplicity, but it is easily generalized. For any $\ell \geq 1$, let $\Delta_t = 1/\ell$ and consider the difference equation

$$x_{t+1} = x_t + \frac{f(x_t)}{\ell}.$$

By running ℓ-many steps of this system in a single time step, i.e. by running it ℓ-times faster, we obtain

$$x_{t+\Delta t} = x_t + \Delta_t f(x_t), \tag{4}$$

whose limit as ℓ goes to ∞ is the ODE system we started with.

We can approximate this limit to an arbitrary degree within the theory. Consider the polynomial $S := \mathbb{R}^n y^{(\mathbb{R}^n)}$, where n is the dimension of the ODE system, and choose $\ell \in \mathbb{N}_{\geq 1}$. Then there is a map of polynomials $S \to \mathbb{R}^n y$, giving an autonomous dynamical system that sends each vector x to $x + f(x)/\ell$. Using Proposition 2.6, we can speed up this dynamical system by a factor of ℓ, to get

$$S \to S^{\circ \ell} \to (\mathbb{R}^n y)^{\circ \ell} \cong \mathbb{R}^{n\ell} y.$$

In other words, this new system outputs ℓ-many n-dimensional vectors in every time step and precisely implements Eq. (4) for arbitrarily small Δ_t.

In order to generalize to input-output systems is easy. Given an arbitrary (context-dependent or input-output) dynamical system $e \colon S \to A$, we can speed it up to get a system $S \to S^{\circ \ell} \to A^{\circ \ell}$, which performs ℓ-many A-operations per time step, each time running the difference equation e.

Combining Example 2.7 and the above notion of strategies, we see control theory as the search for strategies by which to accomplish a goal. We will return to this in Chapter 3.

Monadic Markov decision processes. Markov decision processes (MDPs) are used in reinforcement learning, because they encode a model of non-determinism and reward. MDPs are easy to state in terms of **Poly**; in fact we can generalize the notion of probability and reward to an arbitrary *monad*; we call these monadic MDPs.

Monads—originally a construct from pure category theory—are gaining popularity as a programming paradigm; in particular they have been implemented in Java, Javascript, C#, Haskell, etc. There are monads for all of the following:

1. Probability: functions return probability distributions;
2. Possibility: functions return sets of possible outcomes;
3. Exceptions: functions return either values or exceptions;
4. Rewards: functions return values, together with elements of $\mathbb{R}$.

Monads can sometimes be combined to form new monads; below we will see that MDPs use the monad $S \mapsto \mathrm{Dist}(S \times \mathbb{R})$, involving both probability distributions and reward.

The notion of MDP can be formulated as follows. It has an input language A, a set of states S, and a function

$$\mathit{next} \colon A \times S \to \mathrm{Dist}(S \times \mathbb{R}).$$

Semantically this says that for every input $a : A$ and current state $s : S$, the *next* function returns a joint probability distribution on states and rewards.

We generalize this in two ways:

1. Replace $\mathrm{Dist}(- \times \mathbb{R})$ with an arbitrary monad $\mathfrak{M}$.
2. Allow for partial observability, by including a function $obs \colon S \to B$ for some observation set B. With full observability we'd have $B = A$ and $obs = \mathrm{id}$.

To make this generalization, we use the following fact.

Proposition 2.8. Every monad $\mathfrak{M}$ on **Set** induces a comonad on **Poly**, which we denote $\mathfrak{M}^*$. It sends an arena (P, D) to the arena (P, D') where $D'p \coloneqq \mathfrak{M}(Dp)$ for any $p : P$.

The coKleisli category of the comonad $\mathfrak{M}^*$ is the category for $\mathfrak{M}$-Markov decision processes. In particular, a $\mathfrak{M}$-dynamical system is a map in **Poly** of the form $Sy^S \to \mathfrak{M}^*(P)$ for a set S and arena P. By taking P to be an input-output arena with inputs A and outputs B, the coKleisli morphism is of exactly the kind we wanted:

$$obs \colon S \to B \qquad next \colon A \times S \to \mathfrak{M}(S).$$

Thus we can extend **Poly** by adding effects of any kind—nondeterminism, rewards, exceptions, etc.—as desired.

Summary. The purpose of the preceding paragraphs was to show that the mathematical operations in **Poly** $(+, \times, \otimes, \circ, [-,-]$, etc.) have direct application in terms of dynamical systems, giving direct formalizations of non-determinism, strategies, numerical integration of continuous dynamical systems, wiring diagrams, etc.

But we are only at the beginning phases of our understanding; we next lay out our plans to explore this domain and use it to tackle real scientific questions.

3 Proposed research

Once again, our purpose is to ask and deeply consider fundamental scientific questions regarding the nature of life, interaction, information, communication, etc.

Throughout our lives, and in our work, we consistently see dynamical systems interacting within arenas, whether these be atoms under a microscope, cells in a petri dish, robots on a soccer field, AIs playing Go, organizations hiring talented coders, etc. Such systems get arbitrarily complex, but with a compositional, expressive, and computational formalism, there is no bound to what we can consider; it just requires the effort necessary to explain it precisely.

What we have presented so far in this proposal is only one such setting—albeit one with good potential—in which to work. In our research effort, we will not be bound by a single formalism but instead use whatever tools are available to us. However, it is certainly useful to have a solid place to start, and **Poly** provides that.

Below, we will discuss some of the concrete questions we propose to pursue with suitable funding.

3.1 Mathematics

As discussed above, the category **Poly** has an abundance of formal structures and properties, but there is much to explore.

Porting known concepts. There is a great deal of work on using coalgebras to study dynamical systems, e.g. [AM74], [Jac17], [Has+09], [HKR16], etc. Our setup is more general (see Proposition 1.5) and working completely within **Poly** gives us access to various useful operations that do not seem to have been investigated previously in this context. Still, it is worthwhile to glean as much as we can from these predecessors.

Comonoids in Poly. What makes our dynamical systems really work is the fact that sy^s is a comonoid in $(\mathbf{Poly}, \circ, y)$. It was shown in [AU16] that comonoids in **Poly** are precisely categories. The comonoid sy^s corresponds to a particularly simple category called a contractible groupoid. Richard Garner showed that bimodules between comonoids are exactly parameterized right adjoints in the sense of [Web07], which are exactly *data migration functors* in the sense of [Spi12]. These connections have not been fully understood in the context of dynamical systems, but it will likely lead to some sort of mutual theory of process and data.

String diagrams. Dependent type theory provides a syntax for working with polynomials, however especially when one begins using operations like $\circ$, expressing morphisms and reasoning about their equality becomes complex. Of course this is to be expected: the theory is very expressive and nothing comes for free. Still, it would be useful to have a string diagram language for maps of polynomials. Presumably, this would generalize the static wiring diagrams such as shown in (2).

Double category for generalized lenses. It was shown in [MS20] that the category of polynomial functors with cartesian morphisms is equivalent to the category of Dirichlet functors with cartesian morphisms. In fact there is a double category that includes all of **Poly** and all of **Dir**, the category of Dirichlet functors. One can consider this double category (Myers' *Grothendieck double* construction) for any generalized lens setup as in [Spi19]. If the category **Poly** does turn out to be too restrictive in any way, we will likely try this setting first, as it is a direct generalization.

Generalize beyond sets. The theory of polynomial functors can be formulated in any topos. Perhaps a good topos to work in is smooth spaces, which are a vast generalization of manifolds. It remains to determine whether the dynamical systems applications of the various structures enjoyed by **Poly** still make sense when replacing **Set** by another topos.

3.2 Design environment

We seek to engage as many interested parties as possible in pursuing this work. This includes mathematicians, scientists, engineers, even social scientists, etc. Doing so will be facilitated if we can offer tools that help people think about these complex matters.

Idris, Agda, or Coq implementation. We currently have developed a small library of useful functions in Idris, including all of the structures discussed in Chapter 2 above. We can run the programs and see actual results. However, there is much work to be done.

First, it would be useful to add IO, so that the programs can interface with the outside world. Second, we need to add a number of examples and design patterns, so that users can make learn and create by example.

Domain-specific language. In order to really make this framework available to a larger group of users, we should create a domain-specific language that facilitates the creation of context-dependent interactions of dynamical systems. This language would streamline the syntax and provide visualization tools, better IO interfaces with languages like Simulink, etc.

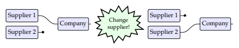
Type search. With operations $+, \times, \otimes, \circ$, etc. at ones disposal, it is quite easy to build new types from an existing library of older ones. How should such a library be organized? Perhaps one needs a dynamical system with a certain interface; is there a way to search the library of existing dynamical systems for one with an interface of type A?

Solvers. To make the design environment most useful, we should implement various solvers, e.g. for numerically integrating a system of ODEs. More interesting would be to have available solvers at least for basic control theory problems.

Operating system that connects to or disconnects from the internet. As a proof of concept, we could develop a basic operating system that works with a keyboard and a monitor, and can connect to internet or not, based on user commands. Of course this is not new, but the hope is that our design environment provides an ergonomic setting in which to build such systems.

3.3 Economics and game theory

We saw on page 16 that strategies, e.g. for game trees, are easily expressed using the composition product $\circ$. The PI was brought to a study of **Poly** by thinking about generalized lenses [Spi19], which are related to work by Ghani, Hedges, and others [Gha+16; Hed17] in the field of open economic game theory. We would like to reunify these two sibling fields.

In economics, supply chains must change dynamically, routing around unforeseeable logistical problems that arise. This sort of situation can be studied within our system, e.g. using context-dependent wiring diagrams such as 

. This is wide-open territory for consideration, and we would like to work with economists to consider it in detail.

3.4 Control

When we admire a great man or woman, when we are astonished at a game-playing robot, what often startles us is their ability to bend reality to their purposes, to make it work for them. This is the intuitive concept of control. The mathematical problem is to design an input-output (or perhaps context-dependent) dynamical system whose outputs keep another system within a stable regime, or more generally keep it close to a given reference signal. One thus proposes their target and aims to hit it.

A control problem for dynamical systems involves formulating a policy for inputs which will drive a given dynamical system through a desired trajectory, or sequence of states. Note that this includes holding the system in a constant state, such as desired by a cruise control system for a vehicle. More generally, one might be interested in the problem of controllability—that is, the existence of control policies for any trajectory chosen from some suitably interesting set—or the yet still more general problem of reachability, i.e. whether it is possible at all to drive the system into a certain state.

Note that the complex interconnectedness of real systems means that any real and useful solution must inherently take into account the possibility of composition—i.e. of interactions with external systems such as obstacles between a robot and its goal—that could frustrate the desired action. In other words, *context matters*. When one does not know the external context, its unknownness must be present as a functorial *variable* in ones model. Otherwise their model will mysteriously break when confronted with the greater reality. This sort of problem arises in synthetic biology, for example.

Denying the greater context by modeling systems in isolation is a fallacy our system naturally avoids. Even a closed box (denoted y) retains a functorial variable for interconnection later. Avoiding the isolation fallacy does not help solve control problems, it only assures that when we do solve them, our answers will be useful even when the system is put in contact with a larger environment. Control problems can be stated in terms of **Poly**, and we want to port existing solution concepts (to the extent they avoid the isolation fallacy) and produce new solutions.

We have said that **Poly** is fully computational, and what this means is that by simply stating a control problem in the mathematics of **Poly**, one has in hand a ready-made simulator for the problem at hand, written in Idris. While simulation can be quite useful, we want to go further by providing tools for tractable, mathematical analysis of control problems. That is, when a class of problems can be tackled in constant time by encompassing them in a mathematical formula, we want methodologies for finding that formula. This is a perennial problem, and one we must confront within our system.

A trade-off between expressivity and computability is inherent to any logic or language. In order to use a framework to solve a given problem, the framework must first be expressive enough to encode it, and then computationally powerful enough to solve it. This is especially critical in active situations where time-to-decision is of paramount importance. We propose **Poly** as an advance for two reasons: first, as already discussed, the category **Poly** is almost singularly rich in structure, which permits powerful reasoning; and second, its compositional translation between 'inner' and 'outer' worlds allows management of approximation.

We hope to collaborate with someone like Jared Culbertson at AFRL on the control theoretic aspect of this research. Let us get a bit more specific.

Computational representation of continuous-time and hybrid dynamical systems. Engineering problems such as the control of aircraft are often in the first instance modeled as continuous-time systems. Any computational solution to the control problem, however, acts digitally; a common way to handle this is by constructing numerical approximations to the continuous-time model. This numerical simulation is often done in an ad hoc manner, with choices made about the approximation hidden, and *external to the model*. What matters for a control problem is simply that the discrete approximation is fine enough.

In our polynomial approach, we must explicitly declare the type of time objects. The structure of **Poly** (in particular the composition product), allows us to also relate different time scales. While other representations of continuous-time systems are possible, and indeed can be used when necessary to bring in tools from differential geometry, this ability to relate time scales opens the option of representing a system as behaving continuously with respect to another as one that is acting on a sufficiently finer timescale. Moreover, the context dependence of our dynamical systems allows natural representation of hybrid dynamical systems as a system with multiple regimes (modeled through an arena's *positions*) that is coupled with a system that pushes it from one regime to another.

As a first step towards practical tools for control, we propose to explore, in our computational design environment, how to construct and control these computationally-aware representations of continuous-time and hybrid systems.

Tractability through approximation. Relatedly, by implementing **Poly** in a dependently-typed language such as Idris, our dynamical systems are built from the ground up in a computationally-aware way. Modeling and computation are not separate in our setup: we do not write programs to numerically approximate our models; the programs *are* the models. The fact that this is built-in allows us to, uniformly within our modeling framework, specify notions of abstraction and approximation. We may relate two models of a system (e.g. a more complex model with richer features, and a simpler model that is more computable) by a morphism describing the approximation, and hence keep track of the detail traded away to improve computation speed.

Tractability through one-sided verification. Compositional methods allow construction of solutions from analyses of simpler parts, which may be done in a distributed manner. Recent work by Baez and Pollard provide one route to computational tractability by providing one-sided guarantees for Markov processes (e.g. methods that can be used to prove a certain state can be reached via analysis of simpler subsystems, but that cannot be used to rule out reachability) through categorical and compositional methods [BP17]. The category **Poly** provides a much more expressive framework in which similar methods could be obtained.

Logics for controllability. Compositional approaches to control are not new; pioneered for example by Willems' behavioral approach. Previous work shows that Willems-style compositional analysis of controllability of dynamical systems benefits from a categorical treatment, for example by permitting a formal logic of wiring diagrams that significantly simplifies proofs of controllability [FSR16]. Like Willems' own work this is based on powerful algebraic techniques for manipulating linear systems, though of course this only goes so far in real-life situations.

On the other hand, approaches such as the template-anchor work of Koditschek and collaborators seek more powerful mathematical underpinning; indeed, recent work in collaboration with Culbertson and others explores the use of categories of hybrid systems and hybrid semiconjugacies [Cul+19]. In addition to the ability to express non-linear and hybrid systems, however, such a category needs enough structure to permit algebraic reasoning. With the wealth of operations detailed in Chapter 2 in mind, we propose that **Poly** may provide the structure needed to take this approach further.

Calculus of variations. In variational calculus, one attempts to find a function that minimizes or maximizes a functional, a real-valued function of functions. It is worth asking whether there is a similar notion for polynomial functors. Given a target behavior of the total system, and given the dynamics of some boxes in a wiring diagram, one can ask the question: what dynamical system can we put in the remaining box to minimize the distance to the target behavior?

Hierarchical planning. Control is often considered a low-level aspect of a general mission, namely how the lowest level actuators respond to given well-designed and stereotyped inputs. We have been using the term 'control' much more broadly. When leaving the world of differential equations and moving into more information-rich worlds, we seek a notion of control that fits at all levels, as well as a way to integrate controllers at these different levels. At the lowest level, we might move actuators at microsecond timescales, whereas at the top level we might make broad-strokes decisions about how to carry out a mission.

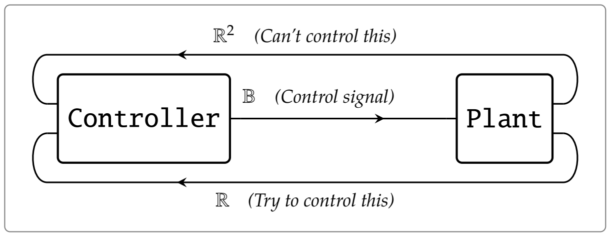
Negotiation between these levels can be formalized category theoretically. However, time-to-decision is a crucial variable that has been left out. Just like a Go-playing AI must decide whether to keep searching for better moves or to play now and keep more clock time for later, timeliness is an important factor in all decision-making. The category **Poly** does have an explicit operation ($\circ$; see page 16) that allow us to consider multiple time-steps at a time. We will investigate whether it or some other structure can be used to fluently pose and answer questions about time-to-decision.

Predictive processing The free-energy minimization approach to prediction and control [Fri10] is gaining traction in a wide variety of applications, e.g. as a "fundamental theory of brain function." The central idea is that higher-level processing units send predictions downward toward the lower-level units (those closer in the pipeline to the sense organs). Rather than reporting the whole scene, the lower-level units report back only the *error*, the difference between the scene and the prediction received. This use of *predictive coding* saves on resources (sending only the error requires fewer bits than sending the whole scene). Moreover, the error signal serves two simultaneous functions: first it corrects the hypotheses of the higher-level units, updating them about the current state of the world. Second, the error can be sent to the actuators as a control signal. The justification for this is quite intriguing and forms what is becoming a large body of literature.

Many believe (e.g. [Mou78; HB07]) that the entire human neocortex is repeatedly performing a single function, and that the differences between the different regions—say the auditory and the visual—are just a reflection of the fact that some are closer to the ears and some are closer to the eyes. The predictive processing and free-energy minimization approach present a plausible theory of what this single function could be.

Thus we will explore whether we can use this notion as a unified approach to control. Our goal is to find a simple enough rule for "what goes in the black box" that we can derive functionality of arbitrary complexity—at least as much as humans achieve—with simple units.

Models revisited When one boxes up the environment and the plant in (2) and does not export the plant state, the resulting diagram is this:[3]


$\mathbb{R}^2$ (Can't control this)
$\mathbb{B}$ (Control signal)
Controller
Plant
$\mathbb{R}$ (Try to control this)

$$\mathbb{B}y^{\mathbb{R}^3} \otimes \mathbb{R}^3 y^{\mathbb{B}} \to y$$

What makes the plant different than the controller is that the controller has a model of the plant, but not vice versa. What then is a *model* in this language? We have already used the term model throughout this document—a model is a morphism in **Poly**—and we can use the same notion here.

In order to understand what it means for the controller to have a model of the plant, we wish to know what the type `CtrlState`—the internal states of the dynamical system `Controller`—should be, as it relates to the interface arena of the plant. Since the states in an arena can be of any type in our implementation, one possibility is to use a type

[3]The wire types, $\mathbb{R}^2, \mathbb{B}, \mathbb{R}$ here can be replaced by any Idris types in the wiring diagram $(cy^{ep} \otimes epy^{c} \to y)$ of the more general control problem.

such as

```
CtrlState = Model (Self (Vect 1000 Double)) Plant.
```

The idea is that the states of the control system could literally be the Idris type of Models in the sense we've been using all along. Here for illustration purposes we imagine that the controller models the plant as having a 1000-dimensional state space.

Again, setting the right state space and dynamics for the controller is a perennial problem and not one that we know how to solve in full generality. However, the self-reflective nature of our language allows us to model models, and hence have dynamical systems that explicitly model other dynamical systems, both in a conventional sense and in the precise sense of morphisms in **Poly**.

3.5 Summary

Thus our story ends where it began: on the subject of models. In order to understand and shape reality, we must model it. Our precise notion of model (morphism in the category **Poly**, shown on page 7) is robust enough to "eat its own dog food". That is, not only are `Model`s the external framework we use to investigate dynamical systems, but the same notion of `Model` can also serve internally as the state space of a dynamical system, e.g. for control.

This self-reflective language is ripe for exploration in a huge number of directions—as we hope to have indicated above—and with it we will consider the deepest scientific questions of our time.

References

[AAG03] Michael Gordon Abbott, Thorsten Altenkirch, and Neil Ghani. “Categories of Containers”. In: *FoSSaCS*. 2003.

[AAG05] Michael Abbott, Thorsten Altenkirch, and Neil Ghani. “Containers: Constructing strictly positive types”. In: *Theoretical Computer Science* 342.1 (2005). Applied Semantics: Selected Topics, pp. 3–27.

[AM74] Michael A Arbib and Ernest G Manes. “Foundations of system theory: decomposable systems”. In: *Automatica* 10.3 (1974), pp. 285–302.

[AU16] Danel Ahman and Tarmo Uustalu. “Directed Containers as Categories”. In: *EPTCS 207, 2016, pp. 89-98* (2016). eprint: arXiv:1604.01187.

[BP17] John C Baez and Blake S Pollard. “A compositional framework for reaction networks”. In: *Reviews in Mathematical Physics* 29.09 (2017).

[BPV06] Aaron Bohannon, Benjamin C Pierce, and Jeffrey A Vaughan. “Relational lenses: a language for updatable views”. In: *Proceedings of the twenty-fifth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*. ACM. 2006, pp. 338–347.

[Cul+19] Jared Culbertson, Paul Gustafson, Daniel E. Koditschek, and Peter F. Stiller. *Formal composition of hybrid systems*. 2019. eprint: arXiv:1911.01267.

[Fri10] Karl Friston. “The free-energy principle: a unified brain theory?” In: *Nature reviews neuroscience* 11.2 (2010), pp. 127–138.

[FSR16] Brendan Fong, Paweł Sobociński, and Paolo Rapisarda. “A categorical approach to open and interconnected dynamical systems”. In: *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science*. 2016, pp. 495–504.

[Gha+16] Neil Ghani, Jules Hedges, Viktor Winschel, and Philipp Zahn. “Compositional game theory”. In: *Proceedings of Logic in Computer Science (LiCS) 2018* (2016).

[GK12] Nicola Gambino and Joachim Kock. “Polynomial functors and polynomial monads”. In: *Mathematical Proceedings of the Cambridge Philosophical Society* 154.1 (Sept. 2012), pp. 153–192.

[Has+09] Ichiro Hasuo, Chris Heunen, Bart Jacobs, and Ana Sokolova. “Coalgebraic components in a many-sorted microcosm”. In: *International Conference on Algebra and Coalgebra in Computer Science*. Springer. 2009, pp. 64–80.

[HB07] Jeff Hawkins and Sandra Blakeslee. *On intelligence: How a new understanding of the brain will lead to the creation of truly intelligent machines*. Macmillan, 2007.

[Hed17] Jules Hedges. *Coherence for lenses and open games*. 2017. eprint: arXiv:1704.02230.

[HKR16] Helle Hvid Hansen, Clemens Kupke, and Jan Rutten. "Stream differential equations: specification formats and solution methods". In: *arXiv preprint arXiv:1609.08367* (2016).

[Jac17] Bart Jacobs. *Introduction to Coalgebra*. Vol. 59. Cambridge University Press, 2017.

[Mou78] Vernon Mountcastle. "An organizing principle for cerebral function: the unit module and the distributed system". In: *The mindful brain* (1978).

[MS20] David Jaz Myers and David I. Spivak. *Dirichlet Functors are Contravariant Polynomial Functors*. 2020. eprint: arXiv:2004.04183.

[SM20] David I. Spivak and David Jaz Myers. *Dirichlet Polynomials form a Topos*. 2020. eprint: arXiv:2003.04827.

[Spi12] David I. Spivak. "Functorial data migration". In: *Information and Computation* 217 (2012), pp. 31–51.

[Spi17] David I. Spivak. "Categories as mathematical models". In: *Categories for the Working Philosopher*. Ed. by Elaine Landry. Oxford University Press, 2017. Chap. 16, pp. 381–401.

[Spi19] David I. Spivak. *Generalized Lens Categories via functors* $\mathcal{C}^{\mathrm{op}} \to \mathsf{Cat}$. 2019. eprint: arXiv:1908.02202.

[ST17] David I Spivak and Joshua Tan. "Nesting of dynamical systems and mode-dependent networks". In: *Journal of Complex Networks* 5.3 (2017), pp. 389–408.

[Web07] Mark Weber. "Familial 2-Functors and Parametric Right Adjoints". In: *Theory and Applications of Categories* 18 (2007), Paper No. 22, 665–732.

Assurances

A Assurances

A.1 Environmental impacts

This research is purely mathematical, and thus it will have no environmental impacts; compliance with environmental statutes and regulations is thereby assured.

A.2 Principle investigator (PI) time

The principle investigator plans to spend 75% of his effort each year of the grant. To carry out a significant fraction of the research proposed above, he would need to hire three to five other researchers, at various levels.

Current Projects and Pending Proposals The PI is starting work at Topos, a new nonprofit research institute, as of January 1, 2020; at the outset it appears he will have no other projects.

A.3 Facilities

Topos provides basic office space and has a budget for books and other resources as requested by its faculty and research staff.

A.4 Special test equipment

None.

A.5 Equipment

None.

A.6 High performance computing availability

Not needed.